HP Records Manager

Software Version: 8.0

.NET SDK

Document Release Date: September 2013
Software Release Date: September 2013
Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements
accompanying such products and services. Nothing herein should be construed as constituting an additional
warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent
with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and
Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard
commercial license.

Copyright Notice

© Copyright 2008-2013 Hewlett-Packard Development Company, L.P.

Trademark Notices

Microsoft®, Windows®, Windows® XP and Windows Vista® are U.S. registered trademarks of Microsoft
Corporation.

Oracle is a registered trademark of Oracle and/or its affiliates.
Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers. HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp
Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the New users - please register link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.
# Table of Contents

**HP Records Manager .Net SDK** ........................................................................................................ 1

This Document ............................................................................................................................................ 1

Binary Compatibility ................................................................................................................................. 1

The Object Browser ................................................................................................................................. 1

Getting started ......................................................................................................................................... 2

Creating a reference to the .Net SDK .................................................................................................... 2

Finding Assemblies at Runtime .............................................................................................................. 2

Programming into HPTRIM ...................................................................................................................... 3

The main objects ..................................................................................................................................... 3

Creating and modifying Records ........................................................................................................... 4

New Records and Electronic Documents ............................................................................................... 9

Locations .................................................................................................................................................. 13

Searching HP TRIM using the .NET SDK ............................................................................................. 15
HP Records Manager .Net SDK

This Document

This document aims to give an overview of programming against HP Records Manager's .Net SDK. It will assume that you are:

1. Familiar with HP Records Manager, it's concepts and usage.
2. Familiar with programming using .Net based languages.

This document will concentrate on getting you started. It's purely about using the HP Records Manager's .Net based API; it will not discuss the HP Service API.

Technically the .Net SDK is an API, but the usage of the terms SDK and API has blurred and they are now used interchangeably.

Binary Compatibility

HP will endeavour to maintain binary compatibility of both of the .Net and COM API's into the future. This means that a program developed against the HP Records Manager 8.0 SDK will not need to be recompiled against future releases of HP Records Manager in order to run. It does NOT mean that:

- Your applications will continue function in the same way. Testing your applications against new releases of HP Records Manager is essential.
- Your code won't get compile errors when recompiled against future releases of HP Records Manager.

The reason for any compile errors or changes in functionality will be found in the SDK release notes which document the history of API changes from version to version of HP Records Manager.

The Object Browser

When developing a HP Records Manager based application you occasionally come to a point where you are not sure what objects, method or properties the .NetSDK has available to perform a specific function. This is where the object browser in your IDE comes in handy. HP has created help strings which provide a simple explanation of all of the objects, methods and properties available in the .NetSDK these can be seen when using the object browser. Using the object browser to search for say ‘rendition’ will give you an idea of what objects, methods and properties have the ‘rendition’ in them, from that point using the help strings allow you to figure out how to do what you are trying to do.
Getting started.

Creating a reference to the .Net SDK

From your .Net IDE you need to find ‘add reference’ either under ‘Project’ on the menu or in the ‘solution explorer’ window and then select the ‘HP TRIM SDK’ component from the .NET tab.

You might want to add ‘using HP.HPTRIM.SDK;’ (C#) or your languages equivalent to your code, so that you no longer need to reference it each time you use one of HP Records Manager object’s methods and properties.

Finding Assemblies at Runtime

.Net uses a process called “Fusion” to locate modules at run time. HP Records Manager however loads all required assemblies and non-managed code libraries independently of Fusion when the method TrimApplication.Initialize() is called. This provides better error reporting when something is wrong.

Currently this logic is implemented:

1. If the TrimApplication.TrimBinariesLoadPath property is set, then search for tsjApi.dll in that location (and don't try any other paths at all).
2. Search for tsjApi.dll in the same folder as HP.HPTRIM.API.dll.
3. Search for tsjApi.dll in the folder specified by the TRIM MSI reg key (HKEY_LOCAL_MACHINE\SOFTWARE\Hewlett-Packard\HP TRIM\MSISettings\INSTALLDIR)
4. Search for tsjApi.dll in parent folders of HP.HPTRIM.API.dll.
5. Search for tsjApi.dll in folders that are on the PATH environment variable.

It is recommended to call TrimApplication.Initialize() explicitly as the first HP Records Manager related call in your code. This results in a simpler error message if something does go wrong.

Example (Calling Initialize):

```csharp
try{
    TrimApplication.Initialize();
    Database db = new Database();
}
catch (Exception e){
    System.Diagnostics.Trace.WriteLine(e.Message);
}
```

If you omit TrimApplication.Initialize(), and there is a subsequent load error, the error will need to be ‘dug out’ through a couple of InnerException's.

Example (retrieving the error through inner exceptions):

```csharp
try{
    //TrimApplication.Initialize();
    Database db = new Database();
    ```
Programming into HPTRIM

The main objects

The .NetSDK has a few of abstract classes from which the rest of its objects descend. They are:

- TrimObject
- TrimMainObject
- TrimChildObject
- TrimChildObjectList
- TrimMainObjectSearch
- TrimPropertySet
- TrimUserOptionSet
- TrimSearchStackItem

Most objects in the .NETSDK have their own constructors and can be created directly.

If while programming an API based application you are unsure about how to perform a specific HP Records Manager function it is often helpful to see how the function is performed in the HP Records Manager Client.

Using the TrimApplication Object

This object has methods and properties which describe and adjust the environment which your application is running in.

Two important methods are:

- Initialize()
  This loads the HP Records Manager runtime environment and reports errors if there are any. It should be called explicitly as the very first call to the HP Records Manager API.
- RuntimeEnvironment = Environments.<enumeration>
  This tells HP Records Manager what the runtime environment of this application is. HP Records Manager will then make changes to its behaviour necessary for that environment.

Using the Database Object

The Database object is one of the main objects managing other HP Records Manager API objects. The Database object is responsible for managing the session connection to a HP Records Manager Workgroup Server.
The Database object is a Creatable object, meaning that you can use the ‘New’ keyword to construct a new instance of it.

Before the actual connection is created you might specify the database to use and the location of the workgroupserver to use.

Example (Database connection):

```csharp
Database objDB = new Database();
objDB.Id = "PD"; //every HP Records Manager Database has a two character id
objDB.WorkgroupServerName = "local";
objDB.Connect(); // throws an exception if the connection fails
```

If the ID and WorkgroupServerName property of the Database are not specified a connection will be made to the users default database when required. If the user has not specified a default database in this situation an exception will be thrown.

Creating and modifying Records

Creating new Records

Before creating a new record you need to load a RecordType object which specifies the type of Record you are creating. After creating a record or changing an existing record you need to save it.

Example (Creating a new record):

```csharp
RecordType RecType = new RecordType(objDB,"Document");
Record objRec = new Record(objDB, RecType);
.
.
// You will need to modify the records properties
.
objRec.Save();
```

Accessing a Record

To read information stored on records in a HP Records Manager database, the API programmer must first determine how to access the records required. If a particular record's internal or external unique identifier is known, the associated record can be accessed directly and efficiently using the Record's constructor. If neither of these unique identifiers is known, it will be necessary to construct a search.

Getting a record by Record Number

Every record in HP Records Manager has a unique Record Number. This follows a pattern defined by the record type and can be manually entered by the user or set to be automatically generated by TRIM. Although the commonly used term is ‘number’, it is more correctly an identifier, as it is a string that may contain alphanumeric characters. This string is accessible through the Record object’s Number property.

The Record Number can be used as the argument to be passed to the Record object’s constructor, which takes a variant for the unique identifier and returns a pointer to the
instantiated Record. For example, if you wish to instantiate the record 2002/0059, you need to use the following statement:

```csharp
Record objRecord = new Record("2002/0059"); // instantiate by number
```

**Note:** HP Records Manager stores the Record Number in two formats, expanded (e.g. “2002/0059”) and compressed (e.g. “02/59”). Either can be passed to the `Record`’s constructor.

**Getting a record by URI**

The Unique Row Identifier or URI of a record is an internal unique number that is transparent to the everyday user of TRIM. It is the primary key on the TSRECORD table in the database and provides an internal unique identifier for every record.

To instantiate a record by its URI, you can pass the numeric URI as the argument to the `Records` constructor, for example:

```csharp
Record objRecord = new Record(130); // instantiate by URI 130
```

Once a record object has been created, the programmer can access properties and call methods on the object. These are discussed in the following subsections.

### Reading Record Data

**Basic Properties**

Most of the metadata directly associated with a record is exposed through properties on the `Record` object. Most properties return primitive data types (strings, numbers or dates) and can be interrogated directly. The meanings of these properties are generally self-evident from their names, but are also given in the object browser. Examples of basic readable properties of a record are:

<table>
<thead>
<tr>
<th>Property</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>“G1997/0770”</td>
</tr>
<tr>
<td>Title</td>
<td>“Greenhouse Journal of Global Warming - Dugong Habitats”</td>
</tr>
<tr>
<td>DateCreated</td>
<td>8/20/1997</td>
</tr>
<tr>
<td>ExternalId</td>
<td>“GJGW 97PB”</td>
</tr>
<tr>
<td>AccessionNbr</td>
<td>5617</td>
</tr>
</tbody>
</table>

**Example (Checking a Records properties):**

```csharp
Record objRec = new Record("G97/770");
if (objRec.AccessionNbr > 5000 & objRec.DateCreated < DateTime.Parse("01/01/2000"))
    MessageBox.Show (objRec.Title, , "Record " + objRec.Number.ToString());
```

**Accessing Related Objects**

Many attributes of a HP Records Manager record represent other objects, such as the `RecordType`, `Classification` and `Container` attributes. These are properties where the data type of the property is an object interface reference.

**Example:**

The following code instantiates a record object (in variable `objRecord`) and then assigns its container to another variable (`objContainer`)
Record objRecord = new Record(“G99/15”);
Record objContainer = objRecord.Container;

Accessing Record Location Information

A HP Records Manager record has various properties concerning related location information. These properties of a Record all return an instantiated Location object:

- CurrentLoc – current location of the record
- HomeLoc – normal location of the record
- OwnerLoc – location of the owner or responsible unit for the record
- AuthorLoc – person who authored the electronic document
- CreatorLoc – person who registered the record in TRIM
- AddresseeLoc – person to whom the record is addressed
- PrimaryContactLoc – the main contact person (or organization) for the record

To access the properties and methods of these location objects, you can create and instantiate them using the following style of code:

Example (Getting a Location from a Record):

Record objRec = new Record(“2002/0059”); // get the record
Location objLoc = objRec.AuthorLoc; // get the author location object
MessageBox.Show (“Author’s name is: ” + objLoc.FormattedName);

Updating Records

So far we have only considered the methods for reading information from records in TRIM. The API also allows you to update HP Records Manager records, either by updating the values of properties on a given record object, or by calling methods on the record.

Updating properties is the simplest way to modify the metadata of a record. You simply assign a new value of the correct data type to the named property of the object. Field-level verification is carried out, and an error will be raised if the property update is invalid (see also the section on ‘Verifying’ below.) For more complicated types of update to a record, you must generally call methods that instruct HP Records Manager to modify the record, based on arguments passed.

Modifying Properties

The simplest way to update data in a HP Records Manager record is to modify the named properties on the Record object. This can only be done on properties that are not marked as read-only. This includes most of the Date properties, certain Location properties (AuthorLoc, AddresseeLoc and OtherLoc) and miscellaneous properties such as External Id, Priority, Accession Number and Foreign Barcode.

Example:

Record objRecord = new Record(30);
objRecord.Title = “New title for this record”;
objRecord.DateDue = DateTime.Today.AddDays(7); //Due in 7 days
objRecord.DatePublished = DateTime.Today;
objRecord.AuthorLoc = objDB.CurrentUser;
Calling update methods

To update other data on a record where read-write properties are not available, you must call a method instead. Update methods often begin with the prefix ‘Set...’ and they include a parameter for the new data value you wish to apply.

```
objRecord.SetCurrentLocation(objMyUnitLoc);
```

In many cases other parameters can be specified that control the behavior of the update:

```
// Set Current location to me, as of yesterday
objRecord.SetAssignee(objDB.CurrentUser(), DateTime.Today.AddDays(-1));
```

Updating properties using SetProperty

To update a record’s properties where the internal identifier of the property is known, you can use the `SetProperty` method. This requires passing the property identifier and a variant containing the data value.

```
// Set the title (property id=3)
objRecord.SetProperty(3, "Barrier Reef manatee population figures");
```

User Defined Fields

HP Records Manager allows user-defined fields to be assigned to records. These user-defined fields cannot be interrogated using normal named properties of the record object. Instead, accessing user-defined fields is carried out using a dedicated object for managing these fields, the FieldDefinition object, and its associated FieldDefinitions collection.

The Record object has a pair of methods for manipulating user-defined fields, `GetUserField` and `SetUserField`. Each method takes a populated FieldDefinition object as a parameter. The field definition object can be instantiated from the Database by passing the field name to the `GetFieldDefinition` method.

Example

```
// Set the field called “User Number 2”
FieldDefinition objUField = new FieldDefinition("User Number 2");
objRecord.SetUserField(objUField, InputBox("User Number 2", "Update", _
    objRecord.GetUserField(objUField)));
```

Verifying and ErrorTrapping

When a record object is modified via the API, there are two levels of verification that must be carried out before the changes can be committed to the database. The first is field-level verification, which checks that the change to an individual property is legal. An example would be that the Date Registered is not in the future. If a property update cannot be carried out because of field-level verification, the method call or property assignment will cause a run-time error to be raised and the update will not be carried out.

The second level of validation is object-level verification (sometimes called cross-field verification.) This checks that the values of all fields on the object are consistent with each other. An example of object-level verification would be that the Date Registered is not earlier than the Date Created. Object-level verification is performed when the object’s Verify or Save methods are called.

The Verify method

The Record object (and every other base object) has a ‘Verify’ method. This can be called to perform object-level verification prior to saving the object. The method returns false if there are any errors in the state of the object, and the error description will be stored in the object’s
ErrorMessage property. If there are no errors, the method returns true and the Verify property (see below) is set to true.

The method contains an optional parameter FailOnWarnings which, if set to true, will cause the Verify method to check for warning conditions as well as error conditions, and to fail if a warning is encountered.

**Example**

```csharp
if (objRecord.Verify(True))
    MessageBox.Show (objRecord.ErrorMessage,, “Verify Failed”);
else
    objRecord.Save();
```

If it is not called explicitly in code, the Verify method will be automatically called before an object is saved (see below) and if verification fails it will not be saved. This ensures that data cannot become corrupted and that business rules are observed when using the API, just as they are for users of the HP Records Manager Client interface.
The Verified property

Base objects also have a 'Verified' Boolean read-only property, which is false whenever the object is instantiated. It is set to true when the Verify method confirms that it is in a legal state to be saved to the database (regardless of warning conditions.)

Trapping Run-Time Errors

It is up to the programmer to determine how they wish to deal with possible errors when updating an object. However, they must be aware that error checking takes place even when directly updating properties, so it will be necessary to provide some error-trapping code to prevent run-time errors being displayed to the user if there is a possibility of errors being raised.

Saving the Record to the Database

All of the update methods and property changes made through the Record interface are only applied to the object in memory. The changes are not committed on the HP Records Manager database until the object is saved.

Calling the Save method on the record object will commit the changes to the database, applying all updates since the object was instantiated (or since it was last saved.) Note that if the record has not been verified, Save will automatically call the Verify method and will only commit the changes if the verification succeeds.

Example

Record objRecord = objDB.GetRecord("G97/770");
objRecord.Title += " plus New Part of Title";
objRecord.DateDue = DateTime.Today.AddDays(5);
objRecord.AuthorLoc = objDB.CurrentUser();
objRecord.Save() ' commit all these changes to the database

New Records and Electronic Documents

Creating a Container File

This scenario describes the general processes for using the SDK to create a record of a generic record type we are calling a 'Container File'. In this and the next scenario (Creating a Document) we are assuming that the reader is familiar with the concept of record types.

While it is up to the Administrator of each HP Records Manager implementation to determine the record types to be used, it is typical to follow a standard records management practise of having at least two record types, one representing Container Files (or Folders) and one representing Documents (the actual names used for the record types may of course vary.) Container Files are usually created and maintained by specialist records managers, as it is generally at this level that classification systems, retention schedules, security, keywords, controlled titling and other records management metadata are applied. Documents, on the other hand, are usually created by end-users, and require little specific metadata other than the identification of the appropriate Container File to which the Document belongs, as all other metadata and context is inherited from the Container.

The general steps for creating a new Container File record are as follows:

1 Instantiate the appropriate Record Type object
2 Instantiate a new Record object of this Type
3 Identify the Classification or Keywords for titling the record (optional)
4 Set the free text title
Creating a Record of a given Type

When creating any record, the Record Type for the new record must be identified. This is passed to the Record object’s constructor along with the database which the new record is to belong to.

```csharp
Create a new Case File record
RecordType objRecType = new RecordType(objDB, “Case File”);
Record objRecord = new Record(objDB, objRecType);
```

**Note:** It is possible to create new Record Types using the SDK; however, this is not recommended as this is generally an Administrator’s function only.

Controlled and Free Text Titling

Titles for Container Files are often subject to controlled vocabulary or classification structures such as a thesaurus or file plan, which give records managers’ greater control over file creation, retrieval and retention. Even when such controlled titling is used, each file will typically also have a ‘free text’ title part. The titling method used is determined by the record type, and is usually set by the HP Records Manager Administrator. Thus a record with Classification titling may have a title such as: “Insurance – Property – Storm damage to Mackay information center”, where the first two terms are generated from a predefined hierarchical classification structure and the remaining part of the title is ‘free text’ describing the specifics of the file. The generated title terms are determined by the classification codes, usually defined as a numerical sequence such as “610/600/”. The free text title is set via the TypedTitle property.

**Example**

```csharp
if (objRecordType.TitlingMethod == tmClassification)
    //Assign classification of 610/600/ = Insurance - Property
    objRecord.Classification = new Classification("610/600/");
    objRecord.TypedTitle = "Storm damage to Mackay information center";
```

Similarly, Thesaurus or Keyword titling allows a file to be titled using either a choice of individual keywords from a controlled list or a specific ‘branch’ of related terms according to a hierarchical structure (similar to a record plan or classification.) A thesaurus-titled file might have a name such as “Administration – Finance – Donations – Bequest from the estate of Lady Marchcroft”

**Example**

```csharp
objRecord.GeneratedTitle = "Administration - Finance - Donations";
objRecord.TypedTitle = "Bequest from the estate of Lady Marchcroft”;
```

Security Levels and Caveats

The security profile of an individual HP Records Manager record is governed by three security controls: a Security Level, a set of zero or more Caveats, and Access Control
Security Levels and Caveats determine the access that a HP Records Manager user has to the metadata of a record. These security specifications are usually applied to Record Types (and inherited by records of each type when they are created) but can be set explicitly on individual records. Every user has a maximum security level and zero or more caveats – in order to access a particular record, the user must have the same or a higher security level and must have all the caveats associated with the record.

Assigning Security Levels and Caveats to a record via the SDK is straightforward. Both the SecurityLevel object and the SecurityCaveat object can be instantiated by full name or by abbreviation. The instantiated SecurityLevel object is assigned to the record’s SecLevel property. Each instantiated SecurityCaveat object can be passed to the record’s AddCaveat method.

Example

```csharp
// Assign “Confidential” level
SecurityLevel objSec = new SecurityLevel(objDB, “CO”);
objRecord.SecLevel = objSec;
// Assign “Research Projects” caveat
SecurityCaveat objCav = new SecurityCaveat(objDB, “RP”);
objRecord.AddCaveat(objCav);
```

Record Locations

Defining relationships between a container file and location objects (people and places) provides additional and useful context for the record.

Unlike record relationships, which can be user-defined, you can only use TRIM’s predefined standard relationship types for record locations.

Record Locations represent actual (in the case of paper and other physical records) or logical (in the case of electronic records) places where a record resides. Every record in HP Records Manager has a property representing it’s Current Location (where the record is now) and another for it’s Home Location (where the record should normally be or where it is to be returned.) There is also a property for Owner Location – the exact meaning of this can vary according to the practises of each HP Records Manager implementation, but normally represents the person or body that is responsible for the record. The Home and Owner location of a record are typically derived from the default values for each Record Type, but all record location properties can be set on creation of a new record or modified later.

The Record object has methods for setting or changing the value of these location properties, which allow the option of specifying the date & time of the change of location (the default is the current time.)

Example

This example sets the record’s Home location to the unit called “Administration”, and the Current location to the connected user.

```csharp
Location objLocation = new Location(objDB, “Administration”);
objRecord.SetHomeLocation(objLocation);
objRecord.SetAssignee(objDB.CurrentUser);
```

Record Contacts

Unlike record locations which tend to be internal units, Record Contacts are more commonly people or organizations that have a direct association with the record, and may be internal or external to the organization. Using the AttachContact method, HP Records Manager allows each contact to be specifically identified as an Author, Addressee, Representative or Client. Other contact relationship types must use the generic type of ‘Other’.
**Example**

This example sets the record’s Representative (and primary contact) to be the connected user, and the Client to be the organization called “Tower”.

```csharp
objRecord.AttachContact(objDB.CurrentUser, ContactType.Representative, true);
Location objLocation = new Location(objDB, “Tower”);
objRecord.AttachContact(objLocation, ContactType.Client, true);
```

### Creating a Document

This scenario describes the general processes for using the SDK to create a record of a generic record type we are calling a ‘Document’.

While Container Files are usually created and maintained by specialist records managers, Documents, on the other hand, are usually created by end-users, and require little specific metadata other than the identification of the appropriate Container File to which the Document belongs, as most other metadata and context is inherited from the Container. A Document record usually consists of an electronic object (the source document, image or other file), a unique identifier (which may be automatically generated by TRIM), a record title and any other metadata required to profile and index the record, and a pointer to the Container File from which the document derives its context.

The general steps for creating a new Document record are as follows:

1. Instantiate the appropriate Record Type object
2. Instantiate a new Record object of this Type
3. Identify the Container File for the document
4. Set the free text title
5. Attach an Electronic file
6. Assign the record’s Author or other contacts (optional)
7. Set Access Control to the electronic document (optional)
8. Assign other metadata or user-defined fields (optional)
9. Save the Record object

### Titling and Numbering

Titling for documents is generally straightforward – free text titling is the norm, and the title simply needs to succinctly describe the document or record. Record numbers may be assigned explicitly or they may be automatically generated – this is configured on the Record Type properties. If the number is explicitly assigned, the number (in expanded format) must be assigned to the LongNumber property (it must be unique or the record will not be saved.)

**Example**

```csharp
objRecord.Title = “Letter from executor regarding disbursements of Lady Marchant’s bequest”;
objRecord.LongNumber = “XK/008934”;
```

### Assigning to a Container

Although it is not compulsory, it is most common that an electronic record is logically assigned to a container file that represents the subject matter, case, client file or other contextual grouping relevant to the document.
To assign a record to a container, the existing container record must be instantiated (by Id or URI) and then passed as an argument to the (contained) record object’s SetContainer method. The method includes a parameter for specifying whether the record is also ‘enclosed in’ the container, i.e. that the current location should reflect that it is with the container.

**Example**

Record objContainer = objDB.GetRecord(“76/915”);
objRecord.SetContainer(objContainer, true);

**Attaching an Electronic Document**

Document records can represent physical paper documents, but mostly they will include an electronic attachment, whether this is a word-processing document, scanned image or other type of file.

To attach an electronic document to a record, the file name and path must be used to instantiate an InputDocument object. This object is then passed as an argument to the record object’s SetDocument method. The method includes parameters for specifying whether this should replace any existing document (or be added as a new revision), whether it should be marked as checked out to the current user, and any comments to be added to the record’s Notes field.

**Example**

InputDocument objDoc = New InputDocument();
objDoc.SetAsFile(“C:\myDocs\ThisFile.doc”);
objRecord.SetDocument(objDoc, false, false, “Created via SDK”);

**Document Author**

*Record Contacts* are HP Records Manager location objects commonly representing people or organizations that have a direct association with the record. The most common type of Contact to be specified for an electronic document is the Author. Although the AttachContact method can be used for this and other contact types, a shortcut is provided through the Author property.

**Example**

This example sets the document’s Author to be the connected user.
objRecord.Author = objDB.CurrentUser;

**Locations**

**Working with Locations**

The Location object is an encapsulation of all properties and methods associated with Persons, Organizations, Positions and Groups. Locations can be identified by name or by URI, and can be selected on other criteria, such as date of birth, nicknames, or membership of a particular organization, role or group.

**Finding a Person by Name**

Although the names of non-persons (Units, Positions and Organizations) must be unique, this is not the case for persons (Staff Names & Contacts). However, HP Records Manager allows you to store a ‘nickname’ for any person, and this can be used as a substitute for a persons name when searching.
To find a particular person by name, you must pass the person’s combined name and title to the `Location` object’s constructor.

**Example 5.1**

```csharp
Location objLoc = new Location("Abbott, Peter (Mr)");
```

If the sub-string does not uniquely identify a location (i.e. there are no matches, or there is more than one match) then a null object will be returned.

**Creating a new Staff Member**

To create a new staff member, you must instantiate a new location by calling the `Location` constructor. You then define the type of the location by assigning a value (in this case `Person`) to the `TypeOfLocation` property. You can then set various properties representing the person’s name, contact details such as telephone numbers and addresses, administrative details such as employee id numbers and so on.

If the new person is to be a HP Records Manager user, then there are login and security details to be provided. You will need to specify the user’s network login id and optionally an expiry date. For the security profile, you are required to either explicitly state the user’s security level (and optionally any Caveats) and a user category, or if role-based security is used you can specify that the user takes the profile of a predefined group or user.

Relationships such as membership of units or reporting lines are created using the `AddRelationship` method and passing parameters for the related location and the relationship type.

Addresses (including electronic addresses such as e-mail or URL) are added by calling the `New` method on the `LocationAddresses` or `LocationEAddresses` collection properties.

**Example**

```csharp
Location objRole = new Location(db, "Project Manager");
Location objLoc = new Location(db);
objLoc.TypeOfLocation = LocationType.Person;

// Name
objLoc.Surname = "Evans";
objLoc.GivenNames = "David";
objLoc.Initials = "D";
objLoc.Honorific = "Mr"

// Personal & Administrative
objLoc.IsWithin = true; // Internal to the org
objLoc.IdNumber = "793906";
objLoc.ReviewDate = DateTime.Today.AddYears(1);
objLoc.DateOfBirth = DateTime.Parse("11/29/1976");
objLoc.PhoneNumber = "555 123496";
objLoc.MobileNumber = "+44 7939 062736";
objLoc.Notes = "Created via SDK";

// Login details
objLoc.CanLogin = true;
objLoc.LoginExpires = DateTime.Today.AddYears(3); // Valid for 3 years
objLoc.LogsInAs = "evans"; // Network login id

// Security
```csharp
if (bRoleSecurity)
    objLoc.UseProfileOf = objRole;

// Confirm & Save
if (objLoc.Verify(true))
{
    objLoc.Save();
    MessageBox.Show (objLoc.FormattedName + " created.");
}
else
    MessageBox.Show (objLoc.ErrorMessage);
```

Searching HP TRIM using the .NET SDK

Searching for TRIM objects

One of the most powerful features of TRIM is the wide range of search criteria that can be applied to select objects from the Database. .NET SDK provides the TrimMainObjectSearch class to provide this functionality. This class allows you to combine a number of search criteria together using boolean logic and then iterate through a resulting set of objects. Note that the rimMainObjectSearch class can only be used to search for objects of a specified type, for instance, you can create a TrimMainObjectSearch to retrieve records. As its name implies, this class can be used to search for any TRIM SDK object that inherits from TrimMainObject.

Constructing a TrimMainObjectSearch

C# Example

```csharp
// Construct a search object to search for records
TrimMainObjectSearch records = new TrimMainObjectSearch(db,
BaseObjectTypes.Record);
```

Specifying the Search Criteria

The TrimMainObjectSearch class provides a number of simple “canned” methods for searching that do not involve any boolean logic. These are simple to use if you have a straightforward query, they include:

- SelectByPrefix
- SelectFavorites
- SelectByUserLabel
- SelectNone
- SelectAll
- SelectByUris
- SelectTopLevels
- SelectThoseWithin

C# Example

```csharp
// Construct a search object to search for all records
```
TrimMainObjectSearch records = new TrimMainObjectSearch(db, BaseObjectTypes.Record);
records.SelectAll();

The second option is to use the string search syntax available in TRIM (refer to the TRIM Help file for examples of the syntax). There are three main components to a string search – the primary search string, a set of filters (these work as if they were combined with the primary search string using a boolean ‘and’) and the sort specification. To create a query in this way, you use the SetSearchString method to specify the primary search string, and then you can optionally specify any filters using SetFilterString and specify a sort using SetSortString. This approach is useful if you want to provide the user with a simple edit control for specifying a search, and is also very useful for web-based applications.

C# Example

// Construct a search object to search for records
TrimMainObjectSearch records = new TrimMainObjectSearch(db, BaseObjectTypes.Record);
records.SetSearchString(“createdOn:this week and assignee:me”);
records.SetFilterString(“type:document”);
records.SetSortString(“createdOn”);

A third option is to work with TrimSearchStackItem objects. The primary search criteria is represented internally as an array of TrimSearchStackItem objects, arranged as a reverse-polish stack to indicate operator precedence. A TrimSearchStackItem can be an operator (TrimSearchOperator) or a search clause (TrimSearchClause). You can build up a query using the internal search stack of the TrimMainObjectSearch, by using such methods as AddSearchClause, And, Or and Not. A read of the Wikipedia article on Reverse Polish notation would be most beneficial for developers unfamiliar with this style of expression. Because filters are all automatically “and-ed” together, they are simply represented as an array of TrimSearchClause items. For sorting, there is a TrimSearchSortItem class and a corresponding array.

C# Example

// Construct a search object to search for records
TrimMainObjectSearch records = new TrimMainObjectSearch(db, BaseObjectTypes.Record);
// set up the individual search clauses
TrimSearchClause recordsCreated = new TrimSearchClause(BaseObjectTypes.Record, db, SearchClauseIds.CreatedOn);
recordsCreated.SetCriteriaFromString(“this week”);
TrimSearchClause recordsAssigned = new TrimSearchClause(BaseObjectTypes.Record, db, SearchClauseIds.Assignee);
recordsAssigned.SetCriteriaFromString(“me”);
// apply the clauses to make a reverse polish stack.
records.AddClause(recordsCreated);
records.AddClause(recordsAssigned);
records.And();
// set up the filter
TrimSearchClause documents = new TrimSearchClause(BaseObjectTypes.Record, db, SearchClauseIds.Type);
recordsCreated.SetCriteriaFromString("document");
// add the filter clause
Records.AddFilter(documents);
// setup a sort item
TrimSearchSortItem sortByDate = new TrimSearchSortItem(SearchClauseIds.CreatedOn);
// add the sort item
Records.AddSort. AddSortItemAscending(sortByDate);

These individual classes provide extra features that allow even more elaborate queries to be built.

Retrieving the results of the search

The TrimMainObjectSearch derives from IEnumerable and so you can use a standard foreach loop to retrieve items that match the selection. In addition, there are two methods that will return the search result as an array of unique object identifiers (corresponds to the Uri property of a TrimMainObject).

C# Example

// Construct a search object to search for records
TrimMainObjectSearch records = new TrimMainObjectSearch(db, BaseObjectTypes.Record);
records.SetSearchString("createdOn:this week and assignee:me")
records.SetFilterString("type:document");
records.SetSortString("createdOn");
// iterate through the records matching the search
foreach ( Record resultRecord in records )
{
    Console.WriteLine(resultRecord.Title);
}

Other search features

Purpose filtering

You can specify that the items in the selection are eventually intended for a specific purpose. This may apply additional hidden filtering to the selection, although not always. As you are iterating through the results, you can use the IsValidForPurpose method to test if the item is suitable for the purpose specified. Each TrimMainObject has an purpose enumeration associated with it – use this enumeration to specify suitable purpose values for the object you are selecting.

Persisting a search

The TrimMainObjectSearch has a SearchAsXML property which will convert the search, filter and sort criteria to an XML string. This string is language-independent and is
therefore useful if you wish to re-use the same search at some later stage, perhaps in a different locale.

**Item matching**

The TrimMainObjectSearch provides an ItemMatches method, which allows you to test whether an existing TrimMainObject matches the search criteria contained within the search.